Benchmarking Hardware and Software Encoder Quality and Performance

Hassene Tmar, Ping-Hao Wu, Cosmin Stejerean, Zhijun Lei, and Ioannis Katsavounidis

Meta Platforms Inc., 1 Hacker Way, Menlo Park, CA 94025, USA

ABSTRACT

In this paper, we present a methodology for benchmarking the coding efficiency and energy efficiency of software and hardware video transcoding implementations. This study builds upon our previous work, which focused on software encoders such as x264, x265, libvpx, vvenc, and SVT-AV1. We have since added a closed-source video software encoder implementation, EVE-VP9, as well as Meta's MSVP VP9 encoder as a hardware representative, and expanded the test set to include a wider variety of test content in our analysis. To ensure a fair comparison between software and hardware encoders, we normalize the video encoding efficiency to energy used in watt-hours. Our proposed test methodology includes a detailed description of the process for measuring compression efficiency and energy consumption. We summarize limitations of our methodology and identify future opportunities for improvement.

Keywords: Video compression, Coding efficiency, Adaptive streaming, Convex hull, Dynamic optimizer, Encoder complexity, Hardware encoder

1. INTRODUCTION

The traditional approach for comparing video codecs has been focused on their compression efficiency using reference software encoders, without considering the complexity of the implementations. However, as the number of video coding standards as well as implementations increases, it has become more complicated to evaluate all of them on a level playing field, considering the vastly different complexity across them. Notable examples include popular open-source software encoder implementations of the AVC,[1] HEVC,[2] VP9,[3] and AV1[4] video coding standards, namely: x264,[5] x265,[6] libvpx,[7] libaom[8] and SVT-AV1.[9] On top of that, modern video encoder implementations typically have different features that can be turned on and off, often offered through a set of presets consisting of curated feature configurations, which provide different levels of trade-off between compression efficiency and computational complexity. Thus there is a growing need to include complexity as an additional dimension in codec comparisons. This is becoming especially critical as practical implementations of video codecs are now available in open-source offerings and are more representative of production-level deployment usages.

In 2018, the Dynamic Optimizer framework[10, 11] was introduced to compare video codecs fairly in different encoding resolutions and quality levels, which better reflects the most popular adaptive streaming use case. In 2019, a compression efficiency and computational complexity analysis was presented,[12] where comparison was made on several known open-source encoder implementations including x264, libvpx, and libaom. The Dynamic Optimizer and software encoding cycle cost were then utilized to compare the performance of software video encoding implementations.[13–15]

However, as video codecs mature and hardware-based implementations become more prevalent, the existing software video codec evaluation methodology, which relies heavily on the amount of CPU cycles consumed during encoding, cannot be directly applied to hardware encoders. Therefore, a new methodology is proposed in this work that combines the Dynamic Optimizer framework with an energy measure (watt-hour) to be able to properly compare hardware and software codecs.

Hassene Tmar: htmar@meta.com
Ping-Hao Wu: npw@meta.com
Cosmin Stejerean: cstejerean@meta.com
Zhijun Lei: ryanlei@meta.com
Ioannis Katsavounidis: ikatsavounidis@meta.com

Applications of Digital Image Processing XLVII, edited by Andrew G. Tescher, Touradj Ebrahimi,
Proc. of SPIE Vol. 13137, 131370W · © 2024 SPIE · 0277-786X · doi: 10.1117/12.3031754

In Sec. 2, we first describe the methodology proposed, and how the axes would be measured in principle. Then we demonstrate how the methodology can be executed practically with detailed configurations, testing and measurement setups in Sec. 3. Challenges, limitations, as well as considerations regarding evaluating the large scale adaptive streaming use case, are then discussed in Sec. 4. Finally we present the executed results in Sec. 5 on selected software encoder implementations as well as the Meta Scalable Video Processor (MSVP),[16,17] the dedicated video ASIC developed by Meta. Finally with the conclusion in Sec. 6.

2. EXPLAINING THE METHODOLOGY

This methodology is focused on an offline framework that would represent codec performance as seen in Video On Demand (VOD) use-cases at Meta as described in.[18] A simplified view of the transcoding pipeline can be found in Fig. 1. A very similar pipeline has been reported by other VOD streaming providers in the past. More details will be discussed in the upcoming section.

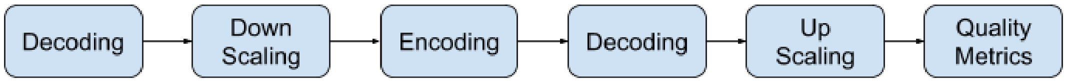

Figure 1. A Simple Transcoding Pipeline for VOD Use Cases

The methodology proposed in this work aims at comparing the encoding compression efficiency of different implementations while taking into consideration the normalized power consumption of the compute component being utilized. As such, the comparison is done across the following two axes: compression efficiency, and the energy usage, which we describe separately in the following. We will first start by comparing the energy efficiency of the core encoding component and we would expand the energy efficiency analysis to the rest of the transcoding components (decoding, scaling, quality measurements).

2.1 Compression efficiency axis

Several approaches on how compression efficiency can be aggregated were presented in our previous work:[14] conventional convex hull approach, combined convex hull approach, and restricted discrete combined convex hull approach. We opt to use the third method: discrete combined convex hull approach, and create an adaptive bitrate (ABR) ladder of 8 lanes targeting quality specific quality levels as measured by VMAF[19] and SSIM.[20] A quick refresh of the construction of these lanes can be found in Fig. 2.

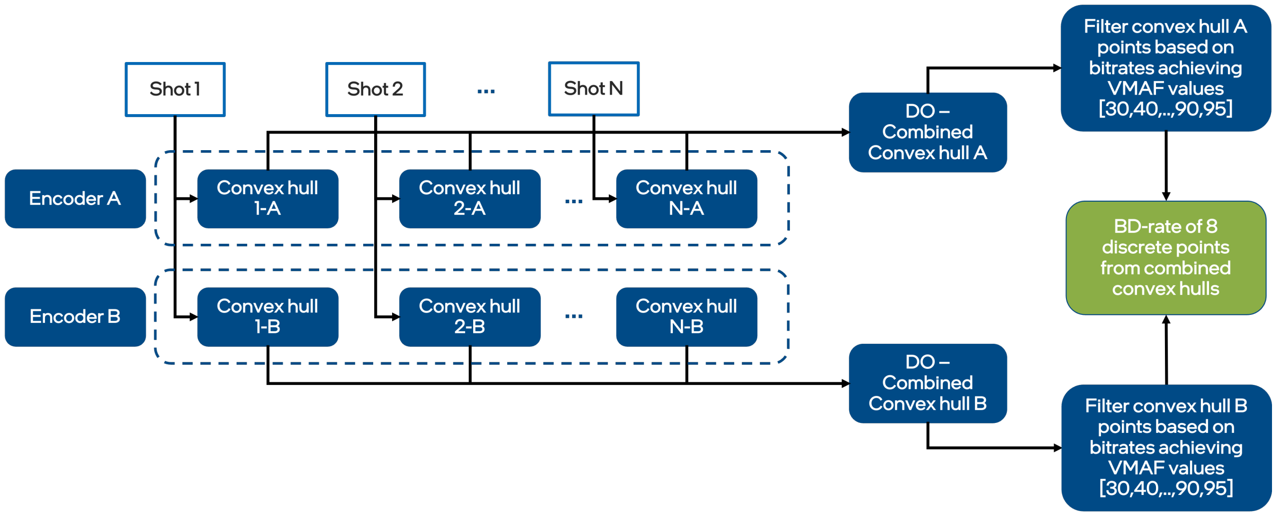
Shot 1
Shot 2
...
Shot N
Encoder A
Encoder B
Convex hull 1-A
Convex hull 2-A
...
Convex hull N-A
Convex hull 1-B
Convex hull 2-B
...
Convex hull N-B
DO – Combined Convex hull A
DO – Combined Convex hull B
Filter convex hull A points based on bitrates achieving VMAF values [30,40,..,90,95]
BD-rate of 8 discrete points from combined convex hulls
Filter convex hull B points based on bitrates achieving VMAF values [30,40,..,90,95]

Figure 2. Combined Convex Hull Construction with Dynamic Optimizer

2.2 Energy usage axis

In recent codec comparisons, we have employed the Linux utility "/usr/bin/time -v" to record the user and system time of the encoding pipeline, when ensuring the system is fully loaded with encoding jobs. This measure has been found to be strongly correlated with energy consumption, as demonstrated previously.[14] Additionally, this metric is neutral and does not rely on implementation-specific reporting.

The time utility calculates the number of cycles used by a process, which essentially tracks the CPU usage of the process. This is quite applicable for software-based implementations. However, for hardware encoding, the processing is performed on specialized hardware components rather than the general purpose CPU. Thus, any cycles captured would not be comparable across hardware and software implementations. Note that even if we could capture the hardware cycle count, it cannot be compared directly with CPUs or with other different hardwares, since they represent different HW architectures, system clock speeds and power consumption per cycle. To address this, we propose to use the power consumption of each component processing the software or hardware implementations. To normalize the comparison, we multiplied the power consumption by the time it takes to run a fixed set of encodings, which is effectively measuring the energy used. This energy measure is also commonly used for electrical bills, electronics consumption factors, and electric vehicle electricity usage. Most importantly, energy consumption is the dominant factor used to determine the operational-expense (OpEx) cost of a service running on Meta's data centers.

3. EXECUTING THE METHODOLOGY

In order to demonstrate how the methodology can be adopted and executed, we have set up a series of tests comparing the compression and energy efficiencies of a variety of publicly available software and hardware implementations. This section outlines the testing setup utilized to construct the 8-lane bitrate ladder along with the BD-rate and power utilization to be measured.

3.1 Content Selection

For this test, we use the following pool of clips to start with:

- El Fuente open source clips processed to 1080p30 resolution and split into 140 shots. Available to download from CDVL.[21]
- A Meta-selected subset of 1080p clips from the YouTube User Generated Content (UGC) test set.[22] This set has square, vertical, and horizontal videos at different frame rates.

We encode all 322 clips with 8 CRF values ranging from 23 - 63 and 5 resolutions (1080p down to 288p) using the SVT-AV1 encoder[13] in preset 8, and compute the VMAF score at original resolutions. K-means clustering was performed on the produced RD (rate-distortion) points. The closest one to the center of each cluster is selected. We thus select a diverse set of contents without having to use the full list.

As a result, we ended up selecting a total of 200 clips split across the following buckets:

- El Fuente: 62 clips
- UGC Square: 49 clips
- UGC Vertical: 50 clips
- UGC Horizontal: 39 clips

Looking at the resulting test set above we have also noticed that the 30fps content represents more than two thirds where the rest is split across 60, 25, and 24 fps. The full distribution by duration is shown in Table. 1.

Table 1. Framerate Distribution of the Selected Video Test Set

FPS	Percentage by duration
60	12%
50	2.8%
30	69%
25	6.7%
24	7.8%
15	1.1%
12	0.6%

3.2 The Transcoding Pipeline

To perform transcoding on the selected video clips, we have followed our previous work.[14] A quick refresher on the pipeline can be in Fig. 3.

The steps are described briefly as follows.

- Group all test sequences that belong to the same class (e.g., 1080p) into subclasses that also have the same, or very similar, frame rate.
- For each subclass, each test sequence is treated as a "shot", part of a longer combined sequence that is a virtual "collage" of all sequences in the subclass.
- After obtaining the convex hull for each "shot", the multiple convex hulls are combined using the approach described in Dynamic Optimizer work.[11] This results in a single rate-distortion curve describing the coding performance over the entire combined sequence.
- From the single rate-distortion curve, 8 operating points are chosen that are as close as possible to the following VMAF quality values: 30, 40, 50, 60, 70, 80, 90, 95. SSIM values are also mapped to the 0-100 scale using the approach described in 23.
- Once a set of 8 ABR lanes have been determined for each encoder implementation and each preset (concrete examples can be found in the later section on executing the methodology), select one particular encoder implementation at a particular preset to be used as the quality reference.
- Compute the Bjontegaard Delta-Rate (BD-rate)[24] difference between each set of lanes representing one encoder, preset and the lanes representing the quality reference. Given that the qualities are usually quite close to the target quality in this case, we can also read the BD-rate difference as a bitrate difference, overhead or savings

3.3 Selecting the Resolutions

We have reduced the resolution selection due to certain hardware encoders not supporting some resolutions used previously. As such the following resolution set has been selected:

[**1920x1080, 1280x720, 960x540, 640x360, 512x288**]

We have opted to pre-downscale the content using the FFmpeg[25] Lanczos filter with alpha parameter set to 5 for all encoders and not utilize the scalars that are available in the hardware System-on-Chip (SoCs) to make sure we restrict the comparison to the encoder's compression efficiency. A sample FFmpeg command line to perform such downscaling is shown here:

```
ffmpeg -y -i input.y4m -sws_flags lanczos+accurate_rnd+full_chroma_int -
    sws_dither none -param0 5 -strict -1 -s:v 1280x720 output.y4m
```

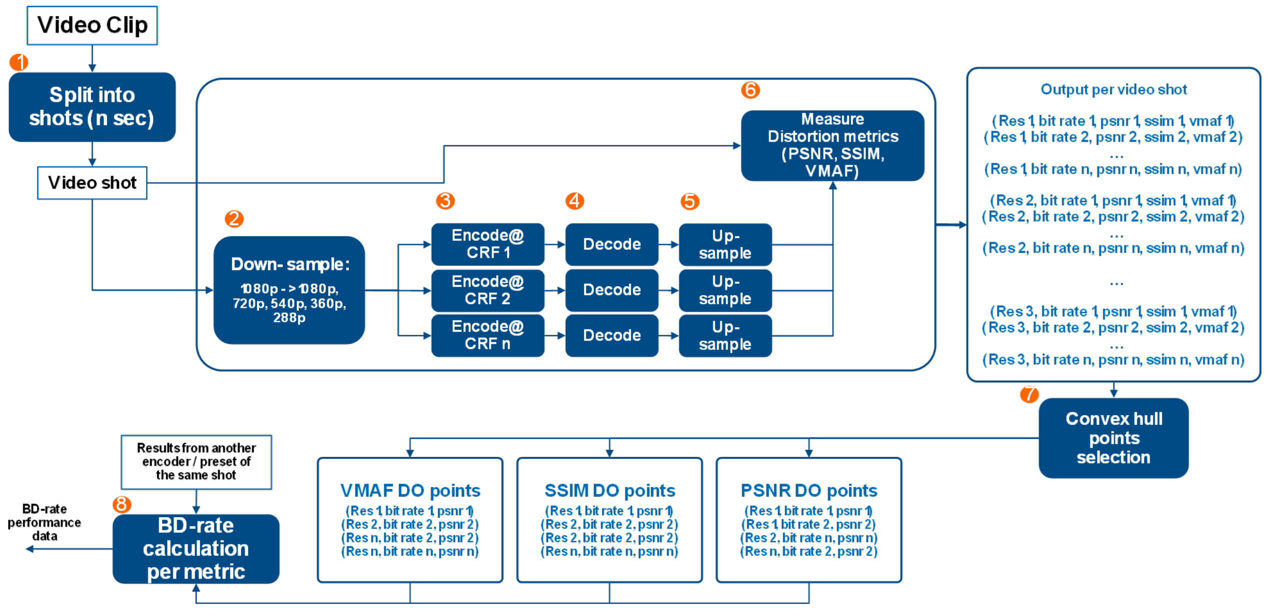
Video Clip
1
Split into shots (n sec)
Video shot
2
Down-sample:
1080p -> 1080p, 720p, 540p, 360p, 288p
3
Encode@ CRF 1
Encode@ CRF 2
Encode@ CRF n
4
Decode
5
Up-sample
6
Measure Distortion metrics (PSNR, SSIM, VMAF)
Output per video shot
(Res 1, bit rate 1, psnr 1, ssim 1, vmaf 1)
(Res 1, bit rate 2, psnr 2, ssim 2, vmaf 2)
(Res 1, bit rate n, psnr n, ssim n, vmaf n)
(Res 2, bit rate 1, psnr 1, ssim 1, vmaf 1)
(Res 2, bit rate 2, psnr 2, ssim 2, vmaf 2)
(Res 2, bit rate n, psnr n, ssim n, vmaf n)
(Res 3, bit rate 1, psnr 1, ssim 1, vmaf 1)
(Res 3, bit rate 2, psnr 2, ssim 2, vmaf 2)
(Res 3, bit rate n, psnr n, ssim n, vmaf n)
7
Convex hull points selection
VMAF DO points
SSIM DO points
PSNR DO points
(Res 1, bit rate 1, psnr 1)
(Res 2, bit rate 2, psnr 2)
(Res n, bit rate 2, psnr 2)
(Res n, bit rate n, psnr n)
Results from another encoder / preset of the same shot
8
BD-rate calculation per metric
BD-rate performance data

Figure 3. Shot Based Encoding Workflow

3.4 Selecting the Encoders

We have chosen a wide selection of software encoder representation representing all codecs, as shown in the following:

- AV1: SVT-AV1 v2.1.0[9]
- VP9: libvpx v1.12.0[7]
- VP9: EVE-VP9 v1.3.52[26]
- AVC: x264 v0.157[5]
- HEVC: x265 v3.6[6]
- VVC: vvenc v1.12.0[27]

Example encoding commands used with each encoder can be found in Appendix A.

For hardware encoders, we have selected the Meta Scalable Video Processor (MSVP) VP9 encoder. A picture of the cards used can be seen in Fig. 4.

3.5 Selecting the Encoding Parameters

We choose the same selected range of CRFs that would align the quality with that achieved by SVT-AV1. The per encoder CRFs are shown as follows for each encoder implementation:

- SVT-AV1: 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63
- libvpx: 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63
- vvenc: 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63
- x265: 19, 21, 25, 27, 31, 33, 37, 39, 43, 45, 47
- x264: 19, 21, 25, 27, 31, 33, 37, 39, 43, 45, 47

Figure 4. Meta Scalable Video Processor (MSVP)

- MSVP VP9: 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63
- EVE-VP9: 92, 108, 124, 140, 156, 172, 188, 204, 220, 236, 252

3.6 Measuring the Compression Efficiency

To assess the compression efficiency, we employed the combined restricted dynamic optimizer method and utilized SVT-AV1 M0 as a quality reference. In this instance, we had a mix of frame rates and a slightly different set of resolutions (1920x1080, 1080x1080, and 1080x1920).

We chose to measure the dynamic optimizer per pixel count bucket, combining 1920x1080 and 1080x1920 into one bucket and considering 1080x1080 separately. This is also done for each class of frame rate. We then combined the final results by adding up the file size per video for all buckets and using a weighted average for the quality to obtain the final number.

For example, calculating the M0 preset results of SVT-AV1 in VMAF would involve the following steps:

- Collect the rescaled metrics across all encodings.
- Process the file through the dynamic optimizer framework to generate 8 bitrate vs quality points for each of the buckets below:

Bucket	**Resolution**	**fps_num**	**fps_den**	**fps**
1080x1080_12_1	1080p square	12	1	12.00
1080x1080_24000_1001	1080p square	24000	1001	23.98
1080x1080_25_1	1080p square	25	1	25.00
1080x1080_30000_1001	1080p square	30000	1001	29.97
1080x1080_50_1	1080p square	50	1	50.00
1080x1080_60000_1001	1080p square	60000	1001	59.94
1920x1080_15_1	1080p vertical or horizontal	15	1	15.00
1920x1080_24000_1001	1080p vertical or horizontal	24000	1001	23.98
1920x1080_25_1	1080p vertical or horizontal	25	1	25.00
1920x1080_30000_1001	1080p vertical or horizontal	30000	1001	29.97
1920x1080_50_1	1080p vertical or horizontal	50	1	50.00
1920x1080_60000_1001	1080p vertical or horizontal	60000	1001	59.94

- Each of the files representing the buckets above will be in the following format (e.g. 29.97 1920x1080):

Bitrate	**VMAF**
83.287	30
126.472	40
193.864	50
304.616	60
495.802	70
872.578	80
1903.656	90
3707.997	95

- Multiply the framecount and divide by the fps of the corresponding bucket to get the filesize for each point:

Bitrate	**VMAF**	**File size (KB)**
83.287	30	38089.14321
126.472	40	57838.67975
193.864	50	88658.65812
304.616	60	139308.2047
495.802	70	226742.1492
872.578	80	399050.8531
1903.656	90	870587.5588
3707.997	95	1695755.985

- Construct the ladder per preset following the two operations:
  - Adding up the resulting file size across all buckets.
  - Taking a weighted average of the quality scores per duration of the bucket vs the whole content.
- We would end up with 8 quality vs. file size points for each preset as shown below for M0 of SVT-AV1:

File size (KB)	**VMAF**
71140	30.23
105469	40.43
159274	50.13
252431	60.03
423250	70.03
788343	79.96
1937022	90.02
4204025	95.01

The above quality vs file size points are used as a reference to compare to any other encoder preset points and measure the BD-rate difference between the two sets of points. We then perform the same operation for each encoder implement and each preset. An example of the VMAF RD-curves comparing the ABR ladders constructed for SVT-AV1 presets M10 to M0 can be found in Fig. 5.

3.7 Measuring and Estimating the Energy Consumption

We first make sure that the platform is utilized as much as possible (either on CPU or hardware card). Doing so in SW pipelines is relatively straightforward by using utilities such as "parallel"[28] which dispatches a desired number of parallel tasks and monitors their completion in order to launch new ones, once some of them finish. For HW pipelines, this is more complicated, since each HW vendor has their own mechanism to interface and dispatch tasks to it, but we in general followed the best practice recommended by such HW vendors. Even with that, there is no practical way to ensure full, 100% utilization of these HW video transcoding cards and more is needed to provide a, hopefully, unified framework that allows users to achieve maximum resource utilization.

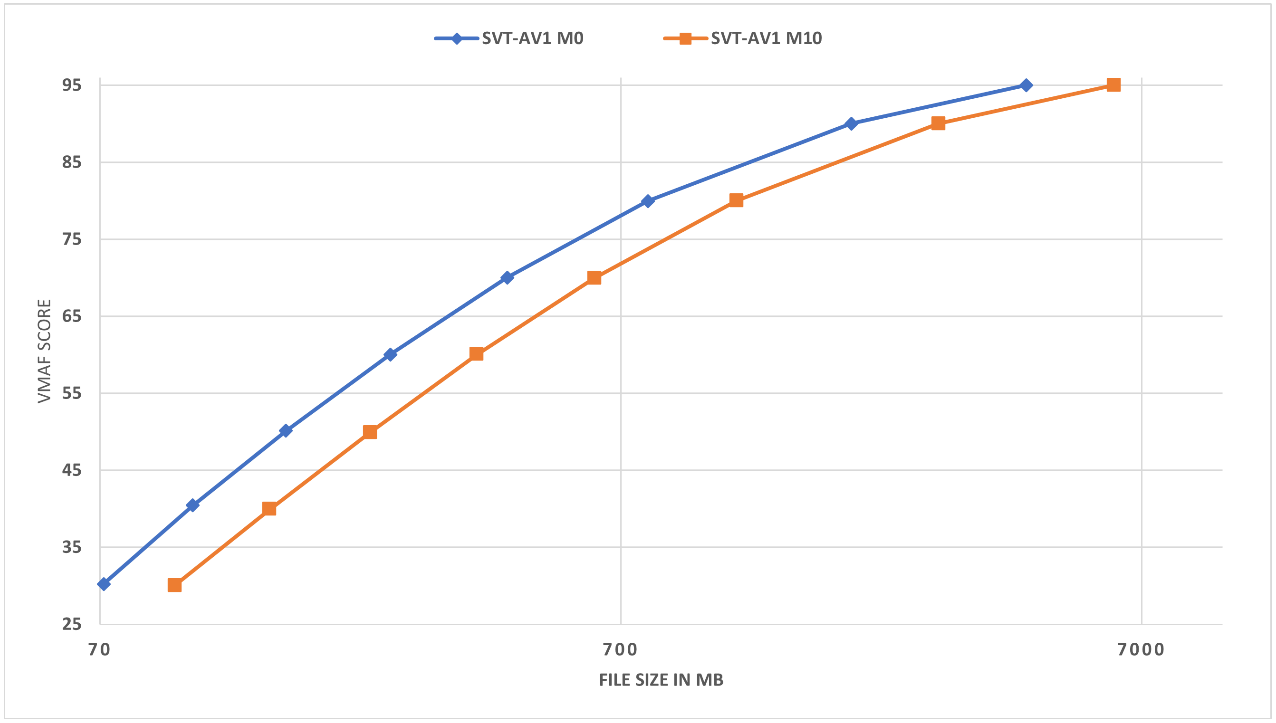

Figure 5. VMAF Rate Distortion curves comparing the final results of SVT-AV1 M10 and M0

The maximum power usage of each platform is used and the time it takes to finish all encoding jobs is measured. We then multiply the watts measure (e.g. 12 Watts for the MSVP HW encoder) by the wall clock to finish the encodings (e.g. 1 hour) to get the watts-hour measure per preset (e.g. 12 watt-hour).

3.7.1 Power numbers utilized for energy usage

In this case, the following power measurements have been used for different platforms:

- To run the hardware encodings MSVP - 12 Watts
- To run the software encodings, we have utilized an AMD Bergamo 96D4 on a 1U server that is rated at 235 Watts

The wall clock measurement was done considering initially only the encoding time and not considering any other part of the pipeline. We have then added the other parts of the pipeline (decoding, scaling, quality metrics) in a separate set of results.

4. CHALLENGES

Several challenges were encountered when we applied the methodology on other publicly available hardware encoders. We have opted not to present the results given these challenges in order not to misrepresent the performance of these offerings. We describe them in three categories: loading the system, accurate and representative power measurement, and a variety of use cases.

4.1 Loading the Card

Unlike the live broadcasting use case, it is common to schedule a wide variety of encoding jobs with different duration and resolutions in a video-on-demand use case. There can be shorter duration as well as lower resolution encodings, both from source or the derived encodings. In our benchmarking on hardware encoders, it appears that some can run into bottlenecks in scheduling and pushing encoding jobs through the card. Specifically, we have seen some performance discrepancies when running exclusively a longer 1080p sequence vs. several shorter sequences with mixed resolutions. Frequent starting up and shutting down the encoding sessions for different video sequences could introduce non-trivial overhead, which prevents us from fully utilizing the hardware engines.

Helper utilities are indeed available to monitor the utilization of different hardware blocks. However, we often need to perform trial-and-error on the more optimal concurrency on different systems, by trying out different concurrent sessions and closely monitoring the utilizations as well as success rate manually. Different resolutions could also need a different number of concurrent sessions to be able to maximize utilization. Encoding failures sometimes can be observed when concurrency is pushed too high.

4.2 Accurate and Representative Power Measurement

It has been a challenge to fairly represent the power consumption or measurement between CPU and hardware in terms of video encoding. There are several factors.

Most commercially available hardware encoders have components other than video encoding. Each system may also contain a different mix of components, which all have a variety of different capabilities. These components can consume power even when sitting idle. In the proposed methodology, we would load the system as much as possible and take the maximum power usage of each platform as the consumption, which does still include the idle components, but there is no easy way to separate them out.

Furthermore, system configurations can quite significantly change the performance. Each hardware encoder in the real world would have to be hosted by a system with CPU, memory, and other components. The choice of these components, as well as the density of the hardware video encoder (i.e., how many such HW accelerator cards are connected on a single host),, which can vary by platform, could affect the normalized power consumption quite noticeably. Even in the case of using a CPU platform, the choice of rest of the system components would also change the consumption. In this benchmarking, we opted to take the entire system for the CPU platform, considering that a hardware encoder card does contain certain general purpose processors as well as on-board memory. While on the other hand, we only look at the maximum power usage of the card itself in this study. We believe that this indeed gives a fairly good representation, given the limitations we mentioned above.

4.3 Covering Use Cases

Major use cases for video encoders are primarily Live and Video-on-demand (VOD). VOD use case at Meta involves full transcoding, high-quality scaling, and quality metrics evaluated at specified viewports.[23] Besides decoding and encoding, such a pipeline includes high quality downscaler, high quality upscaler, as well as diverse sets of quality metrics. We have found that many existing solutions have different quality levels in the downscaler; do not necessarily include a high quality up-scaler; and often times only with limited selection of quality metrics. Furthermore, any one of the different available hardware blocks could become the bottleneck, which highly depends on the use case.

With that in mind, we opted to present the execution of the methodology with a set of encoder-only experiments to eliminate any other factors, such as different length filter taps in downscaler. However, we believe that to accurately capture and evaluate large scale adaptive streaming use cases, the full pipeline should be taken into account.

5. RESULTS

The executed results are presented in this section with graphs. Each color on the graph represents a particular codec implementation, and each point corresponds to a particular preset for that implementation. The y-axis shows the average VMAF/SSIM BD-Rate of each preset compared to SVT-AV1 M0. The x-axis (in logarithmic scale) is generated by multiplying the wall clock time required to complete all tests, by the power consumption of the system, 235W for the CPU workloads and 12W for the MSVP card. The results in Fig. 6 show that there is indeed an advantage of hardware video encoder over the state-of-the-art software implementations and systems. That is, at similar compression efficiency, hardware video encoders consume much less energy, even

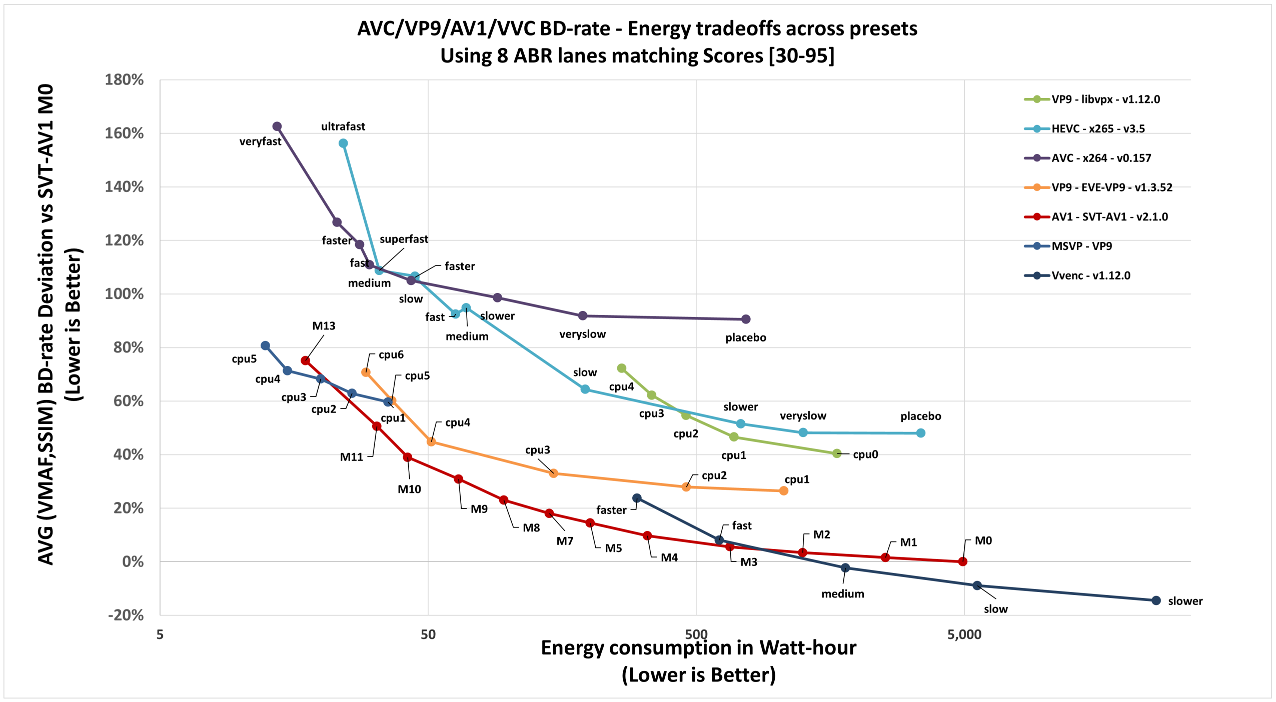

Figure 6. BD-rate vs Energy Consumption Trade Offs across Different Encoders

when comparing VP9, an earlier generation codec, to the latest AV1. We do see though that the energy savings advantage is reduced as we go to higher quality presets of the MSVP VP9 encoder.

Furthermore, as stated in the previous section, in real world large scale deployment, the full pipeline must be accounted for. This includes the cost of the full transcoding and quality metrics pipeline: decoding, downscaling, encoding, upscaling, and quality metrics calculation. In this case, the transcoding pipeline including upscaling and VMAF calculations measured each 16 frames can be done in MSVP without much impact to its energy usage.[17] While in the software pipeline, the components other than encoding can consume a larger portion of the pipeline especially when operating at a high speed encoding preset.

Following the same methodology described above and assuming a full utilization of the CPU, we have measured the wall clock of each of the transcoding components, other than encoding, by taking the following steps:

- Encode the input video with x264 medium with default settings at CRF 20 to have a compressed input. A sample command line is shown here:

  ```
  ./ffmpeg -i "$file" -c:v libx264 -crf 20 -preset medium "$output_file"
  ```

- We first measure the wall clock of decoding and downscaling the input bitstream. We found this component of the pipeline has the smallest energy consuming step in the software pipeline, representing $<1\%$ of the total energy usage of the pipeline, as it is done only once for 1080p input streams.
- We then measure the wall clock of decoding the resulting bitstreams using the available decoders in FFmpeg, upscaling and calling the libvmaf library with the additional parameter "subsample=16". We found that these components can take a small portion of the energy consumption of the pipeline: $\sim 1\%$ when encoding with SVT-AV1 M0, or it can add up to $>60\%$ of the pipeline when encoding with a much faster encoding preset such as M13.
- We then sum up all of these as a fixed cost to all presets and software implementations on the curve, and update the graph accordingly.

Considering the full cost of the pipeline, we now see an energy savings of the hardware implementation that goes anywhere from ∼2x to ∼5x. The results are shown in Fig. 7.

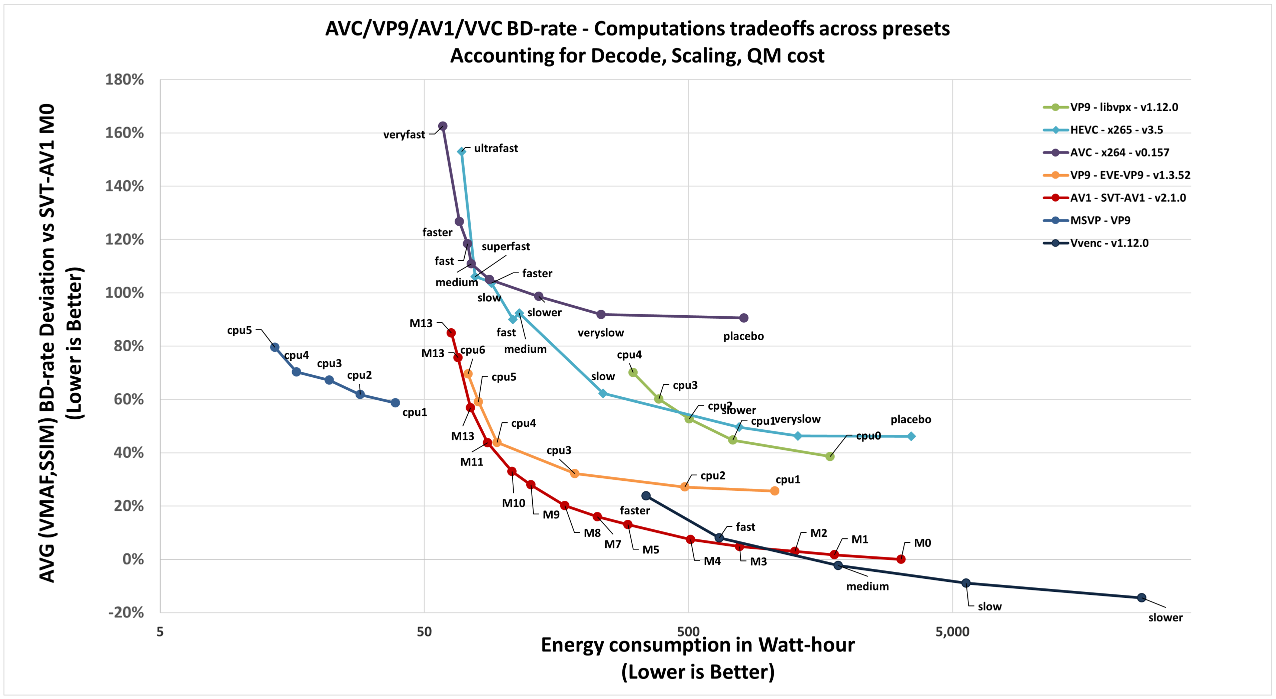

Figure 7. BD-rate vs Energy Consumption Trade Offs across Different Transcoding Pipelines

6. CONCLUSION

In this work, we present a methodology of evaluating and comparing software and hardware video encoders, on different platforms, in a level playing field. This extends our previous work on comparing software implementations fairly and representative of large scale production use cases.

We demonstrate how this methodology can be executed in practice by employing it on various software encoders as well as a dedicated video ASIC. We also present what the results would look like when the full transcoding pipeline is taken into consideration, for better matching large scale deployment use cases. The results further show that, with optimized hardware design, a previous generation video codecs can still outperform software implementations of the latest generation codecs on the energy vs. coding efficiency plane. We highlight how the integration of high-quality downscaling, upscaling and state-of-the-art video quality metrics (such as SSIM and VMAF) in a HW video transcoding card can bring significant savings over SW solutions, where many times those additional components account for a large portion of complexity/power requirements.

Several challenges on evaluating across different platforms have also been presented. With the exception of Meta's MSVP HW transcoder, which was designed with the Video-On-Demand use case in mind, other HW solutions lack a proper (and ideally unified) interface that allows for maximum loading of HW encoders in a reliable way. We believe that the proposed methodology provides a framework to evaluate as well as aid in developing the next generation of video codec implementations, whether they are in software or hardware.

APPENDIX A. SAMPLE ENCODING COMMAND

- x264:

```
./ffmpeg -y -i <input> -g 301 -keyint_min 301 -crf <crf> -c:v libx264 -
    threads 1 -preset <2,9> -tune psnr -x264-params scenecut=0 -f h264 <
    output>
```

- x265:

```
./ffmpeg -y -i <input> -g 301 -keyint_min 301 -crf <crf> -threads 1 -c:v
    libx265 -preset <0,9> -tune psnr -x265-params pools=1:wpp=0:no-info=1:
    scenecut=0 -f hevc <output>
```

- vvenc:

```
./ffmpeg -i <input> -vcodec libvvenc -preset <0,4> -vvenc-params
    internalbitdepth=8:intraperiod=301:qp=23:threads=1:accessunitdelimiter
    =0 -f rawvideo <output>
```

- libvpx:

```
./vpxenc  --ivf --codec=vp9  --tile-columns=0 --arnr-maxframes=7 --arnr-
    strength=5 --aq-mode=0 --bias-pct=100 --minsection-pct=1 --maxsection-
    pct=10000 --i420 --min-q=0 --frame-parallel=0 --min-gf-interval=4 --
    max-gf-interval=16 --verbose   --passes=2 --end-usage=q  --lag-in-
    frames=25 --auto-alt-ref=6  --threads=1  --profile=0  --kf-min-dist
    =301 --kf-max-dist=301 --cq-level=<crf> --cpu-used=<0,4>  -o  <output>
       <input>
```

- EVE-VP9:

```
./ffmpeg -y -i <input> -g 301 -c:v libeve-vp9 -speed <1,6> -threads 1 -
    tune  psnr -cqf <crf> -keyint_min 301 -reorderconstraint none -f ivf <
    output>
```

- SVT-AV1:

```
./ffmpeg -y  -i <input> -crf <crf> -preset <0,13> -g 301 -threads 1 -c:v
    libsvtav1  -f ivf  -svtav1-params lp=1 <output>
```

- MSVP VP9:

```
./ffmpeg -y -init_hw_device fbv:/dev/tenjin0 -i <input> -filter_complex "
    [0:v]format=nv12,hwupload,format_fbv=f=tile,scale_fbv=iw:ih,split=2[
    main][tmp];[tmp]scale_fbv=hme=1[hme];[main][hme]splice_fbv[out]" -map
    '[out]' -c:v vp9_fbv -noautoscale -vsync -1 -vp9_alt_ref 1 -preset
    hq_vod -gop_size 300 -crf <crf> -b:v 0 -f ivf <output>
```

ACKNOWLEDGMENTS

The authors of the paper would like to thank P'sao Pollard-Flamand from the Intel team for the help developing and updating the transcoding scripts and contributing them to the open source under the AOM testing resources repository.[29]

REFERENCES

[1] ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), "Advanced video coding for generic audiovisual services," (2003).
[2] ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), "High efficiency video coding," (2013).
[3] Grange, A., de Rivaz, P., and Hunt, J., "VP9 bitstream and decoding process specification," (2016).
[4] de Rivaz, P. and Haughton, J., "AV1 bitstream & decoding process specification," (2019).
[5] "x264, code repository - open-source AVC encoder software." link: https://github.com/mirror/x264.

[6] "x265, code repository - open-source HEVC encoder software." link: https://bitbucket.org/multicoreware/x265_git.

[7] "libvpx, code repository - open-source VP9 encoder/decoder software." link: https://github.com/webmproject/libvpx.

[8] "libaom, code repository - open-source AV1 encoder/decoder software." link: https://aomedia.googlesource.com/aom.

[9] "SVT-AV1, code repository - scalable video technology for av1 (svt-av1 encoder)." link: https://gitlab.com/AOMediaCodec/SVT-AV1.

[10] Katsavounidis, I., "The NETFLIX tech blog: Dynamic optimizer — a perceptual video encoding optimization framework." https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f (Mar. 2018).

[11] Katsavounidis, I. and Guo, L., "Video codec comparison using the dynamic optimizer framework," in [*Applications of Digital Image Processing XLI*], Tescher, A. G., ed., **10752**, 266 – 281, International Society for Optics and Photonics, SPIE (2018).

[12] Katsavounidis, I., "Encoding vs. compute efficiency in video coding, video @Scale 2019." link: https://atscaleconference.com/events/video-scale-2019/ (Sept. 2019).

[13] Kossentini, F., Guermazi, H., Mahdi, N., Nouira, C., Naghdinezhad, A., Tmar, H., Khlif, O., Worth, P., and Amara, F. B., "The SVT-AV1 encoder: overview, features and speed-quality tradeoffs," in [*Applications of Digital Image Processing XLIII*], Tescher, A. G. and Ebrahimi, T., eds., **11510**, 1151021, International Society for Optics and Photonics, SPIE (2020).

[14] Wu, P.-H., Katsavounidis, I., Lei, Z., Ronca, D., Tmar, H., Abdelkafi, O., Cheung, C., Amara, F. B., and Kossentini, F., "Towards much better SVT-AV1 quality-cycles tradeoffs for VOD applications," in [*Applications of Digital Image Processing XLIV*], Tescher, A. G. and Ebrahimi, T., eds., **11842**, 118420T, International Society for Optics and Photonics, SPIE (2021).

[15] Cobianchi, G., Meardi, G., Poularakis, S., Walisiewicz, A., Abdelkafi, O., Amara, F. B., Kossentini, F., Stejerean, C., and Tmar, H., "Enhancing SVT-AV1 with LCEVC to improve quality-cycles trade-offs and enhance sustainability of VOD transcoding," in [*Applications of Digital Image Processing XLV*], Tescher, A. G. and Ebrahimi, T., eds., **12226**, 122260S, International Society for Optics and Photonics, SPIE (2022).

[16] Katsavounidis, I., Lalgudi, H., Lan, J., Reddy, H., and Chen, Y., "MSVP: Meta's first ASIC for video transcoding." https://ai.meta.com/blog/meta-scalable-video-processor-MSVP/ (May 2023).

[17] Reddy, H. M., Chen, Y., Lan, J., Katsavounidis, I., Anandharengan, B., Lalgudi, H. G., Alaparthi, S., Hua, G., Chuang, H.-C., Wu, P.-H., Lei, Z., Mastro, A., Petersen, C., Chaudhari, G., Prakash, P., Regunathan, S., Reddy, S., Venkatapuram, P., Rao, V., Noru, K., Bjorlin, A., Zeile, M., Lewis, A., Singh, A., Sunil, A., Chen, C.-C., Lin, C.-F., Chen, C., Sundar, D. P., Jayaraman, D., Ucar, H., Li, H., Singh, J., Liu, J. C. C., Rachamreddy, K. R., Sriadibhatla, K., Datla, K., Berg, L. V. D., Feng, L., Jampani, P., Moola, R., Mallya, R., Jha, S., Pan, S., Srinivasan, S., Vaduganathan, V., Zha, X., Wang, Z., Sengottuvel, A. K., Alluri, B., Oshin, B., Kanumetta, C., Sahin, E., Athaide, J. M., Wu, J., Kurapati, K. C., Manthati, K., Thottempudi, K., Chennamsetti, R. R., Jagannath, K. R., Arvapalli, S., Kala, T., Wang, T., Chopda, P., Gandhi, K., Ramesh, A., Gupta, R., Fadnavis, S., Qassoud, A., Friedt, C., Li, F., Gao, H., Lee, J., Dixit, M., Ugaji, S., Karuturi, T., Xie, X., Narasimha, A., Jakka, B., Dodds, B., Yang, J., Skandakumaran, K., Modi, M., Modi, P., Stejerean, C., Ronca, D., Wang, H., Pham, N., Lu, L., Shen, H., Ning, J., Narayanan, K., Chen, L., Avidan, N., Arnold, W., Xu, F., Patil, G., Balan, V., and Grandhi, S. D., "Efficient video processing at scale using MSVP," in [*Applications of Digital Image Processing XLVI*], Tescher, A. G. and Ebrahimi, T., eds., **12674**, 1267414, International Society for Optics and Photonics, SPIE (2023).

[18] "How meta brought av1 to reels." link: https://engineering.fb.com/2023/02/21/video-engineering/av1-codec-facebook-instagram-reels/.

[19] Li, Z., Aaron, A., Katsavounidis, I., Moorthy, A., and Manohara, M., "The NETFLIX tech blog: Toward a practical perceptual video quality metric." link: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652 (2016).

[20] Zhou Wang, Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P., "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing* **13**(4), 600–612 (2004).

[21] "The Consumer Digital Video Library." link: https://www.cdvl.org/.

[22] "YouTube-UGC Dataset." link: https://media.withyoutube.com/.

[23] Regunathan, S. L., Wang, H., Zhang, Y., Liu, Y. R., Wolstencroft, D., Reddy, S., Stejerean, C., Gandhi, S., Chen, M., Sethi, P., Puntambekar, A., Coward, M., and Katsavounidis, I., "Efficient measurement of quality at scale in Facebook video ecosystem," in [*Applications of Digital Image Processing XLIII*], Tescher, A. G. and Ebrahimi, T., eds., **11510**, 69 – 80, International Society for Optics and Photonics, SPIE (2020).

[24] Bjontegaard, G., "Calculation of average PSNR differences between RD-curves," *ITU-T Q.6/SG16 VCEG 13th meeting* (2001).

[25] "FFmpeg." link: https://ffmpeg.org/.

[26] "EVE-VP9 encoder." link: https://www.twoorioles.com/eve-vp9.

[27] "VVenC, the fraunhofer versatile video encoder - code repository." link: https://github.com/fraunhoferhhi/vvenc.

[28] Tange, O., "Gnu parallel 20230822 ('chandrayaan')," (Aug. 2023). GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.

[29] "Testing Resources for the AOMedia Codec Projects." link:https://gitlab.com/AOMediaCodec/aom-testing/-/tree/master/scripts/content-adaptive-streaming-pipeline-scripts.